



PATENT ABSTRACTS OF JAPAN

(11) Publication number: **2002229445 A**(43) Date of publication of application: **14.08.02**

(51) Int. Cl.

G09C 1/00
G06F 7/72
(21) Application number: **2001021128**(22) Date of filing: **30.01.01**(71) Applicant: **MITSUBISHI ELECTRIC**
CORPMITSUBISHI ELECTRIC
SYSTEM LSI DESIGN CORP(72) Inventor: **ASAMI KAZUO**(54) **MODULATOR EXPONENT DEVICE**

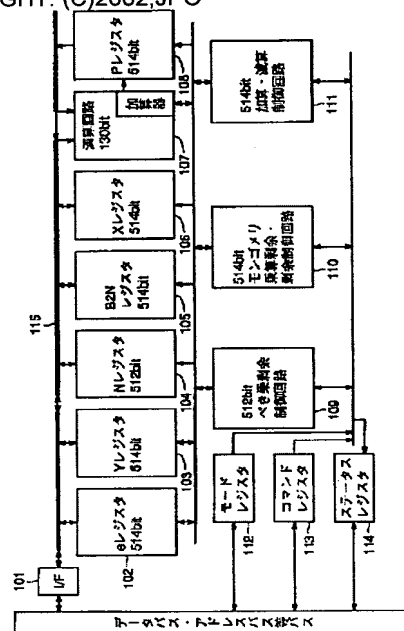
(57) Abstract:

PROBLEM TO BE SOLVED: To provide a modulator exponent device which can perform high-speed processing.

SOLUTION: The modulator exponent circuit includes an I/F (interface) circuit 101 which is an interface to an external bus, an e register 102 which holds a key e, a Y register 103 which holds a multiplier Y for making Montgomery transformation, an N register 104 which holds a key N, a B2N register 105 which holds the value of $2B+N$ to be performed in computation of the Montgomery transformation, an X register 106 which holds plaintext X, an arithmetic circuit 107 for making computation for the purpose of encryption and decryption, a P register 108 which holds the result P of the computation, a power-residue control circuit 109 which is ought to play the role as a state machine in executing the modulator exponent, a Montgomery multiplication residue and residue control circuit 110 which plays the role as a state machine in executing Montgomery modulator exponent and remainder computation and an addition and subtraction control circuit 111 which performs the computation control of

addition and subtraction.

COPYRIGHT: (C)2002,JPO



(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号
特開2002-229445
(P2002-229445A)

(43) 公開日 平成14年8月14日 (2002.8.14)

(51) Int.Cl. ⁷	識別記号	F I	テ-コード* (参考)
G 0 9 C 1/00	6 2 0	G 0 9 C 1/00	6 2 0 A 5 J 1 0 4
G 0 6 F 7/72		G 0 6 F 7/72	

審査請求 未請求 請求項の数6 O L (全 10 頁)

(21) 出願番号 特願2001-21128(P2001-21128)

(22) 出願日 平成13年1月30日 (2001.1.30)

(71) 出願人 000006013

三菱電機株式会社
東京都千代田区丸の内二丁目2番3号

(71) 出願人 391024515

三菱電機システムエル・エス・アイ・デザ
イン株式会社
兵庫県伊丹市中央3丁目1番17号

(72) 発明者 朝見 和生

兵庫県伊丹市中央3丁目1番17号 三菱電
機システムエル・エス・アイ・デザイン株
式会社内

(74) 代理人 100064746

弁理士 深見 久郎 (外4名)

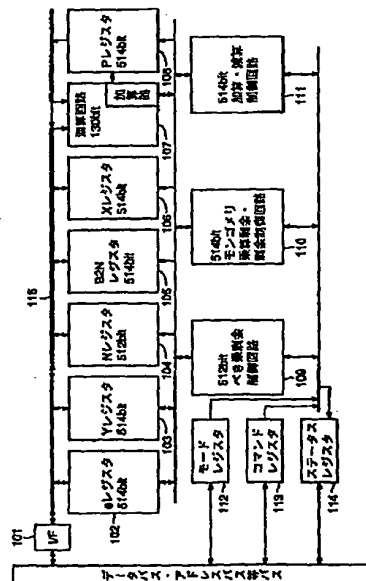
Fターム(参考) 5J104 AA22 JA28 NA18

(54) 【発明の名称】 べき乗剰余演算器

(57) 【要約】

【課題】 高速処理が可能なべき乗剰余演算器を提供す
る。

【解決手段】 べき乗剰余演算回路は、外部バスとのイ
ンタフェースである I/F (インタフェース) 回路10
1と、鍵eを保持するeレジスタ102と、モンゴメリ
変換をする乗数Yを保持するYレジスタ103と、鍵N
を保持するNレジスタ104と、モンゴメリ変換の演算
時に行なう $2B+N$ の値を保持する $B2N$ レジスタ10
5と、平文Xを保持するXレジスタ106と、暗号化お
よび復号化のための演算を行なう演算回路107と、演
算結果Pを保持するPレジスタ108と、べき乗剰余演
算実行時のステートマシンとしての役割を果たすべき乗
剰余制御回路109と、モンゴメリ乗剰余演算と剰余
演算との実行時のステートマシンとしての役割を果たす
モンゴメリ乗剰余・剰余制御回路110と、加算およ
び減算の演算制御を行なう加算・減算制御回路111と
を含む。



【特許請求の範囲】

【請求項1】 モンゴメリ乗算剰余演算を行なう際の一方の引数を2倍した値と剰余の法とを加算した値を保持するレジスタと、

前記レジスタに接続され、前記レジスタに保持された値を参照して、モンゴメリ乗算剰余演算を実行するためのモンゴメリ乗算剰余演算実行手段と、

前記モンゴメリ乗算剰余演算実行手段に接続され、前記モンゴメリ乗算剰余演算実行手段との間でデータのやり取りを行ない、べき乗剰余演算を実行するためのべき乗剰余演算実行手段とを含む、べき乗剰余演算器。

【請求項2】 前記べき乗剰余演算実行手段は、2進数表現されたべき指数の各ビットの値に関わらず、モンゴメリ乗算剰余演算実行手段においてモンゴメリ乗算剰余演算を実行し、べき乗剰余演算を実行する、請求項1に記載のべき乗剰余演算器。

【請求項3】 さらに、べき乗剰余演算実行手段における演算モードを保持するためのモードレジスタを含み、前記べき乗剰余演算実行手段は、前記モードレジスタに保持された値に基づいて、2進数表現されたべき指数の各ビットの値に基づいたモンゴメリ乗算剰余演算を実行するかどうかを判断し、モンゴメリ乗算剰余演算を実行する、請求項1または2に記載のべき乗剰余演算器。

【請求項4】 前記べき乗剰余演算実行手段は、2進数表現されたべき指数の各ビットの値に基づいて、モンゴメリ乗算剰余演算を実行するかどうかを判断し、モンゴメリ乗算剰余演算を実行する、請求項1～3のいずれかに記載のべき乗剰余演算器。

【請求項5】 前記べき乗剰余演算実行手段は、2進数のビット列を加算する加算器を含み、前記加算器は、2進数のビット列を所定ビットごとに分割し、分割後のビット列同士で加算を行なう複数のサブ加算器を含む、請求項1～4のいずれかに記載のべき乗剰余演算器。

【請求項6】 さらに、前記モンゴメリ乗算剰余演算実行手段および前記べき乗剰余演算実行手段に接続され、演算の一部を取出して実行するための手段を含む、請求項1～5のいずれかに記載のべき乗剰余演算器。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、情報通信ネットワーク、交通、金融、医療、流通等の分野において使用される情報の暗号化技術および復号化技術で使用されるべき乗剰余演算器に関し、特に、モンゴメリ(Montgomery)のアルゴリズムを用いてべき乗剰余演算を行なうべき乗剰余演算器に関する。

【0002】

【従来の技術】情報通信技術の発展に伴い、情報ネットワーク上のセキュリティの確保(データの盗用や破壊を*

$$N = p \cdot q \quad \dots (4)$$

*防止すること)が重要視されるようになってきている。

そのため、情報の暗号化技術および復号化技術が採用されることが多く、その適用分野は単なる情報通信分野に留まらず、交通、金融、医療、流通等の身近な分野にまで広がりがつつある。この種の暗号化技術および復号化技術には、高度なセキュリティを単純な原理によって実現できることが要求される。

【0003】まず、この種の技術の理解を容易にするため、情報の暗号化・復号化についての概略を説明する。暗号の世界においては、“非対称暗号アルゴリズム”が質的に優れている。非対称暗号アルゴリズムとは、暗号化鍵と復号化鍵とが異なり、そのいずれか一方から他方が“容易に計算できない”暗号アルゴリズムをいう。この非対称暗号アルゴリズムの代表的なものに、べき乗剰余演算(ある数Xを何回も乗算してNで割った余りをとる計算)を用いるタイプのRSA(Rivest-Shamir-Adleman scheme)暗号がある。

【0004】RSA暗号を生成するには、次式(1)のべき乗剰余演算の形式が基本として使用される。式

(1)は、X'をNで割ったときの余りを求めることを意味する。また、式(1)において、Xは暗号化(復号化)の対象となる明文、YおよびNは暗号化(復号化)のための鍵(キー)である。

【0005】 $X' \bmod N \quad \dots (1)$

このべき乗剰余演算を用いることにより、情報の暗号化および復号化が容易に実行され、かつX、Y、Nのオペランドビット長を長くすることで、各鍵の解読を困難にすることができる。

【0006】しかし、オペランドビット長を長くすると、べき乗剰余演算に長時間を要することになる。そこで、オペランドビット長が長いべき乗剰余演算をいかに短時間に終了させるかがポイントとなる。

【0007】次に、RSA暗号を例に取り、べき乗剰余演算を使用した暗号化処理および復号化処理について説明する。

【0008】[RSA暗号の暗号化および復号化について]

(1) RSA暗号の暗号化には、次式(2)が用いられる。

【0009】 $C = M^e \bmod N \quad \dots (2)$

復号化には、次式(3)が用いられる。

【0010】 $M = C^d \bmod N \quad \dots (3)$

ここで、Mは暗号化の対象となる明文、Cは暗号化された明文すなわち暗号文である。また、式(2)におけるeおよびNは暗号化鍵、式(3)におけるdおよびNは復号化鍵である。また、以下の式(4)および式(5)の関係が予め与えられている。

【0011】

$$1 \equiv e \cdot d \pmod{\text{LCM}(p-1, q-1)} \quad \dots (5)$$

ここで、「 \equiv 」は、左辺と右辺とが相似であることを意味し、「LCM」は、最小公倍数を意味する。また p と q とは互いに素な整数である。なお、 e および N は公開鍵であり、 d 、 p および q は秘密鍵である。

【0012】式(4)および式(5)は、ともに暗号アルゴリズムにおけるべき乗剰余演算の数値の条件を定義している。式(4)は、 N は互いに素な大きな素数 p および q の積であることを示している。 p および q はともに奇数なので、当然 N は奇数でなければならない。式(5)は、式(4)で示した p および q からそれぞれ1を減じた値同士の最小公倍数で、 e および d の積 $e \cdot d$ を当該最小公倍数で割ったときの余りが1になることを

(フロー1)

begin

A=1

for i=k-1 to 0

begin

A=A²modN ... (6)

If eⁱ=1 then A=A·MmodN ... (7)

end

end

Aに格納された値が求めたいべき乗剰余演算の解になる。

【0016】以上のように演算の基本は、式(6)および式(7)に示すように乗算と除算(mod算)である。乗算は、初期値を1とするAの値に対してA×AまたはA×Mを行なう部分である。除算は、各々の乗算で得られた値に対してmodN(Nで割ったときの余りを求める演算)を行なう部分である。この“乗算と除算”(A×AmodN、A×MmodN)を1対の演算として、“e”のビット値に従って繰返し演算が行なわれる。すなわち、“e”の最上位ビットから最下位ビットまでの各ビットの内容によって“乗算と除算”が行なわれる。

【0017】べき乗剰余演算は、基本となる剰余演算(mod算)を繰返し行なうことで解が得られることを示したが、この繰返し回数自体は、たかだか数百〜数千回であるので、ソフトウェアによる処理でも十分に対応できる。しかし、この剰余演算自体すなわち除算をハードウェアによって実行するためには、大規模な演算回路と複雑な処理手順とが必要とされる。このため改善が望まれていた。通常、 e 、 d 、 M 、 N などは1024ビット程度の大きな整数が用いられているので、高速指数計算法を使用しても1回のRSA演算で平均1500回程度の多重精度乗算と剰余算とを行なわなければならない。特に剰余算は、近似法、剰余テーブル方式、モンゴメリのアルゴリズム等、多くの高速化手法が提案されている。

【0018】このような、RSA暗号に代表される公開鍵暗号の多くで利用される、べき乗剰余演算を高速に処

理している。

【0013】式(4)および式(5)の条件に基づき、平文Mは式(2)を用いて暗号化され、また暗号化された平文M(暗号文C)は式(3)を用いて復号化される。

【0014】[べき乗剰余演算の演算方法について]次に暗号化・復号化で使用される、べき乗剰余演算の演算方法を説明する。 $A=M^e \pmod{N}$ のべき乗剰余演算は、整数eの2進数展開を $e=e^{k-1} \dots e^1 e^0$ として、以下のフロー1に示す反復平方積法を用いることにより実行される。

【0015】

理するためには、1回当たりの剰余算の高速化が要求される。モンゴメリのアルゴリズムは、剰余算を高速処理するアルゴリズムである。特に、乗算剰余演算においては、除算をビットシフトなどで簡略化できるため、公開鍵暗号(RSA暗号等)で用いられるべき乗剰余演算を高速処理することができるという特徴がある。

【0019】一方、中国人の剰余定理により、合成数を法とする演算は合成数を構成する互いに素な因数を法とする演算から計算できる。これを1024ビット長RSA暗号処理に適用すると、実際に必要なハードウェアとしては、1024ビット長の法Nによるべき乗剰余演算回路ではなく、512ビット長の整数(ここではpおよびqに相当する)を法とする演算回路のみでよい。このためハードウェアの小型化に繋がる。

【0020】べき乗剰余演算は、基本となる剰余演算(mod算)を実行する手順が非常に複雑であるため、演算回路が大規模化してしまうことを上述した。そこで、モンゴメリは、剰余演算(mod算)を先のような一般的な方法で行なわずに、“乗算”と簡単なビット列処理とで行なうことによって解を得る仕組みを提案している。以下にモンゴメリが提案している手法について簡単に説明する。

【0021】[モンゴメリのアルゴリズム] 剰余演算の高速化を実現する一手法であるモンゴメリのアルゴリズムについて説明する。

【0022】モンゴメリのアルゴリズムは、剰余の法N($N>1$)と、剰余の法Nと互いに素である基数R($R>N$)とを用いると、被剰余数をTとした場合、 TR^{-1}

modNの計算が基数Rによる除算のみで行なえる性質を利用している。これにより、Nによる除算を用いることなく剰余計算を行なうことができる。ここで、N、R、 R^{-1} およびTは整数である。被剰余数Tは $0 \leq T < R \cdot N$ を満たす数である。 R^{-1} は剰余の法Nの上での基数Rの逆数である。またここでさらに、 $R \cdot R^{-1} - N \cdot N' = 1$ ($0 \leq R^{-1} < N$, $0 \leq N' < R$) の関係を満たす整数 N' を考えることができる。さらに、この基数Rに2のべき乗数を使用した場合、基数Rによる除算をシフト操作に置き換えることができる。このため、 $T \rightarrow TR^{-1} \bmod N$ (被剰余数をTとした場合の $TR^{-1} \bmod N$) の計算の高速処理が可能となる。

【0023】次にアルゴリズム1として、 $T \rightarrow TR^{-1} \bmod N$ のアルゴリズムMR(T)を示す。ただし、アルゴリズム1において $(T+m \cdot N)/R$ は必ず割り切れることが証明されている。

【0024】(アルゴリズム1) $T \rightarrow TR^{-1} \bmod N$ のアルゴリズム $Y = MR(T)$ は次のように表わされる。

【0025】

$$M = (T \bmod R) \cdot N' \bmod R \quad \dots (8)$$

$$Y = Rr \quad (Rr = R^2 \bmod N \quad (R = 2^{k+2}))$$

$$X = M$$

$$X = MR(X, Y) \quad \dots (10)$$

$$Y = MR(1 \cdot Y) \quad \dots (11)$$

for j=k to 1

$$\text{if } e_j == 1 \text{ then } Y = MR(X \cdot Y) \quad \dots (12)$$

$$\text{if } j > 1 \text{ then } Y = MR(X \cdot Y) \quad \dots (13)$$

end for

$$Y = MR(1 \cdot Y) \quad \dots (14)$$

$$Y = Y \bmod N \quad \dots (15)$$

ここで、 $MR(X \cdot Y)$ と $MR(Y \cdot X)$ とは等しく、 e_j はキーeのjビット目を表わす。512ビット長の整数の場合 $k=512$ となり、512ビットのべき乗剰余演算は514ビットのモンゴメリ乗算剰余演算と512ビット剰余演算とにより実現できる。

$$W = 2^d$$

$$N0' = N' \bmod W$$

$$P = 0$$

for j=0 to k

$$M = (P \bmod W) \cdot N0' \quad \dots (16)$$

$$P = ((P + (A \bmod W) \cdot B \cdot W + M \cdot N) / W) \bmod 2^k \quad \dots (17)$$

$$A = A / W \quad \dots (18)$$

End

dは自然数であり、ハードウェアに依存する数である。このようにして、モンゴメリ乗算剰余演算結果Pを求めることができる。d=1の基数2の逐次計算で514ビ

$$N0' = N' \bmod 2$$

$$P = 0$$

for j=0 to 514

$$Y = (T + m \cdot N) / R \quad \dots (9)$$

if $Y \geq N$ then $Y = Y - N$

$Y < N$ then return Y

1回のMRでは、剰余 $T \bmod N$ ではなく $TR^{-1} \bmod N$ が求められるだけである。よって、剰余 $T \bmod N$ を求めるためには、次に示すようにMR(T)と予め求めておいた $R^2 \bmod N$ との積で、再びMR演算を行なえばよい。

【0026】

$$MR(MR(T) \cdot (R^2 \bmod N))$$

$$= (TR^{-1} \bmod N) \cdot (R^2 \bmod N) \cdot R^{-1} \bmod N$$

$$= TR^{-1} \cdot R^2 \cdot R^{-1} \bmod N$$

$$= T \bmod N$$

このようにして剰余 $T \bmod N$ を求めることができる。

【0027】以上のことにより、モンゴメリ法による乗算剰余演算を使用して、これをべき乗剰余演算の反復平方積法(繰返し2乗法)で実現するアルゴリズムを下記に示す。鍵eの上位ビットから検索し、鍵のビットの値が1の場合には、 $MR(X \cdot Y)$ のモンゴメリ乗算剰余演算を行なう。

【0028】

【0029】また、ハードウェアとして実装するのに最適な基数Wの逐次計算でモンゴメリ乗算剰余演算結果 $P = MR(B \cdot A)$ を求めると、以下のようになる。

【0030】

ットモンゴメリ乗算剰余演算結果 $P = MR(B \cdot A)$ を求めると、以下のようになる。

【0031】

$$M = (P \bmod 2) \cdot N0' \quad \dots (19)$$

$$P = ((P + (A \bmod 2) \cdot B \cdot 2 + M \cdot N) / 2) \bmod 2^{514} \quad \dots (20)$$

$$A = A / 2 \quad \dots (21)$$

end

以上のように、べき乗剰余演算を実現するために、ハードウェアで512ビット長のべき乗剰余演算にモンゴメリ法を使用し、ソフトウェアで中国人の剰余定理を利用した処理を使用することが従来採用されている。ハードウェアへの実装方式は複数通りあり、実際に様々な方式が採用されていると思われる。

【0032】

【発明が解決しようとする課題】しかし、従来の回路では、図8に示すような手順で処理が行なわれている。すなわち、512ビット長のべき乗剰余演算にモンゴメリ法を使用した回路を実装したハードウェアを用い、式(10)～式(18)を式そのままに実行している。たとえば、 $e_j = 0$ のときは式(12)をスキップしている。一方、式(17)においては毎回すべての計算を実行している。そのため複雑な処理手順が必要とされ、より一層の高速化が望まれている。また、回路規模が小さくて済み、LSI (Large Scale Integration) 化に適した回路が必要となっているので、演算処理をできるだけ簡略化し全体の計算量を削減し、処理速度の向上を図ることが必要である。

【0033】本発明は上述の課題を解決するためになされたもので、その目的は、高速処理が可能なべき乗剰余演算器を提供することである。

【0034】

【課題を解決するための手段】本発明のある局面に従うべき乗剰余演算器は、モンゴメリ乗算剰余演算を行なう際の一方の引数を2倍した値と剰余の法とを加算した値を保持するレジスタと、レジスタに接続され、レジスタに保持された値を参照して、モンゴメリ乗算剰余演算を実行するためのモンゴメリ乗算剰余演算実行手段と、モンゴメリ乗算剰余演算実行手段に接続され、モンゴメリ乗算剰余演算実行手段との間でデータのやり取りを行ない、べき乗剰余演算を実行するためのべき乗剰余演算実行手段とを含む。

【0035】モンゴメリ乗算剰余演算に頻繁に用いられる値をレジスタに保持することにより、モンゴメリ乗算剰余演算を高速に実行することができる。

【0036】好ましくは、べき乗剰余演算実行手段は、2進数表現されたべき指数の各ビットの値に関わらず、モンゴメリ乗算剰余演算実行手段においてモンゴメリ乗算剰余演算を実行し、べき乗剰余演算を実行する。

【0037】べき指数の各ビットの値に関わらず、常にモンゴメリ乗算剰余演算が実行される。このため、べき乗剰余演算器を暗号化装置および復号化装置に用いた場合であっても、タイミング攻撃に対する耐性を確保する

ことができる。

【0038】さらに好ましくは、べき乗剰余演算器は、さらに、べき乗剰余演算実行手段における演算モードを保持するためのモードレジスタを含み、べき乗剰余演算実行手段は、モードレジスタに保持された値に基づいて、2進数表現されたべき指数の各ビットの値に基づいたモンゴメリ乗算剰余演算を実行するか否かを判断し、モンゴメリ乗算剰余演算を実行する。

【0039】モードレジスタに保持された値に基づいて、2進数表現されたべき指数の各ビットの値に基づいたモンゴメリ乗算剰余演算を実行するか否かを判断する。このため、テスト時には、2進数表現されたべき指数の各ビットの値に基づいたモンゴメリ乗算剰余演算を実行するようにし、実際使用する際には、べき指数の各ビットの値に関わらず、常にモンゴメリ乗算剰余演算を実行するようにすれば、テスト時間を短縮でき、かつタイミング攻撃に対する耐性を確保することができる。

【0040】さらに好ましくは、べき乗剰余演算実行手段は、2進数表現されたべき指数の各ビットの値に基づいて、モンゴメリ乗算剰余演算を実行するか否かを判断し、モンゴメリ乗算剰余演算を実行する。

【0041】2進数表現されたべき指数の各ビットの値に基づいて、モンゴメリ乗算剰余演算を実行するか否かを判断し、モンゴメリ乗算剰余演算を実行する。このため、ビットの値として1が出現するまでの間は、モンゴメリ乗算剰余演算の結果が予めわかっている場合がある。このような場合に、処理をスキップすることにより、高速に処理することができる。

【0042】さらに好ましくは、べき乗剰余演算実行手段は、2進数のビット列を加算する加算器を含み、加算器は、2進数のビット列を所定ビットごとに分割し、分割後のビット列同士で加算を行なう複数のサブ加算器を含む。

【0043】加算器をサブ加算器に分割してパイプライン処理を実行することにより、高速に加算を行なうことができる。このため、べき乗剰余演算を高速に実行することが可能になる。

【0044】さらに好ましくは、べき乗剰余演算器は、さらに、モンゴメリ乗算剰余演算実行手段およびべき乗剰余演算実行手段に接続され、演算の一部を取出して実行するための手段を含む。

【0045】べき乗剰余演算時に行なう各種演算を取出して実行することにより、様々な種類の暗号化処理を実現できるようになる。

【0046】

【発明の実施の形態】【実施の形態1】

【べき乗剰余演算回路の全体構成】図1を参照して、本発明の実施の形態1に係るべき乗剰余演算回路は、外部バスとのインタフェースであるI/F（インタフェース）回路101と、鍵eを保持するeレジスタ102と、モンゴメリ変換をする乗数Yを保持するYレジスタ103と、鍵Nを保持するNレジスタ104と、モンゴメリ変換の演算時に行なう $2B+N$ の値を保持するB2Nレジスタ105と、明文Xを保持するXレジスタ106と、暗号化および復号化のための演算を行なう演算回路107と、演算結果Pを保持するPレジスタ108と、べき乗剰余演算実行時のステートマシンとしての役割を果たすべき乗剰余制御回路109とを含む。

【0047】べき乗剰余演算回路は、さらに、モンゴメリ乗算剰余演算と剰余演算との実行時のステートマシンとしての役割を果たすモンゴメリ乗算剰余・剰余制御回路110と、加算および減算の演算制御を行なう加算・減算制御回路111と、各種モードを保持するモードレジスタ112と、コマンドを保持するコマンドレジスタ113と、ステータスを保持するステータスレジスタ114と、各種レジスタと演算回路107間でのデータのやり取りを行なうための内部バス115とを含む。

【0048】べき乗剰余演算を行なうにあたり、高速化を実現するためにモンゴメリ法による乗算剰余演算を使用しているが、そのモンゴメリ法の演算 $[P=MR(B \cdot A)]$ 行なう最初に $[2B+N]$ なる計算を行ない、その結果をB2Nレジスタ105に格納する。

【0049】モンゴメリ法の演算 $MR(B \cdot A)$ の動作について説明する。ここで、 $B=X$ 、 $A=Y$ の場合を考える。まず、 $[2B+N]$ の計算を行なう。 $[2B+N]$ の演算は、以下のようにして行なわれる。演算回路107は、Nレジスタ104に保持された値を0と加算し、Pレジスタ108に記憶する。演算回路107は、Xレジスタ106に保持された値を2倍し、その値とPレジスタ108に保持された値とを加算し、Pレジスタ108に書込む。モンゴメリ乗算剰余・剰余制御回路110は、Pレジスタ108に保持された値をB2Nレジスタ105に書込む。次に、式(19)、(20)および(21)の演算を515回繰返し実行する。

【0050】モンゴメリ乗算剰余演算中の式(20)を計算時、 $A \bmod 2 == 1$ および $M == 1$ の場合はB2Nレジスタ105からデータを読み出し、内部バス115を介し読出されたデータとPレジスタ108に記憶された値とを演算回路107にて加算する。また、 $A \bmod 2 == 0$ および $M == 1$ の場合はNレジスタ104からデータを読み出し演算回路107にてPレジスタ108に記憶

された値と加算する。 $A \bmod 2 == 0$ および $M == 0$ の場合は $[0+P]$ の加算を実行する。0は内部バス115を流れるデータを0にするなどして生成される。eレジスタ102は鍵e、Yレジスタ103はY、Nレジスタ104は法N、Xレジスタ106はXを保持するレジスタで、Pレジスタ108はモンゴメリ乗算剰余演算の式中式(19)および式(20)に出てくる値Pを保持するレジスタである。

【0051】【演算回路107に設けられた加算器の構成】図2を参照して、演算回路107内に設けられたパイプライン処理を実行する加算器について説明する。

【0052】この加算器は、モンゴメリ乗算剰余演算を行なう加算器の構成を複数のサブ加算器に分割し、加算器のキャリーの段数を少なくした構成としている。加算器は、サブ加算器内のキャリー回路(SubC回路)727~741と、サブ加算器内でキャリーが発生する可能性を示すキャリー回路(LookC回路)742~752と、サブ加算器間でキャリーを伝播させるためのキャリー回路(MainC回路)720および726と、加算結果を得るためのキャリー回路(SlaveC回路)716~719および721~725と、実際に加算処理を行なうアダー回路(adder)701~715とを含む。

【0053】キャリー回路(LookC)742~752は、下位のサブ加算器でキャリーが発生したと仮定して、ビット同士の加算を行ない、キャリーが発生する可能性を判断する。

【0054】一例として、15ビットのパイプライン処理を実行する加算器について説明する。

【0055】15ビットの場合、下位ビットから4+5+6=15の段数が適当とする。まず、4段のサブ加算器内では普通に加算が行なわれ、キャリーが検出され、加算結果が得られる。5段のサブ加算器では、4段のサブ加算器からのキャリーを含めてSlaveC回路とadder回路により5段の加算結果を得る。また、別途同時に5段内だけのキャリーをSubC回路で検出する。またLookC回路は下位のビットからキャリーが発生したと仮定してキャリー信号を検出する。

【0056】そして4段のサブ加算器からのキャリー信号で次の6段のサブ加算器へのキャリーをSubC回路735とLookC回路745からの信号どちらにするかをMainC回路720が選択する。同様に6段のサブ加算器でも加算結果を得る。

【0057】たとえば、4段のサブ加算器において「1111+1110」の加算が行なわれる場合を想定する。各ビットでの演算は以下になる。

【0058】

0ビット	$0+1=1$	キャリーなし
1ビット	$1+1=0$	キャリーあり
2ビット	$1+1+1$ (1ビットからのキャリー)	$=1$ キャリーあり
3ビット	$1+1+1$ (2ビットからのキャリー)	$=1$ キャリーあり

このとき、SubC回路730からは「キャリーあり」の信号がMainC回路720に与えられる。よって、このときは、MainC回路720は、下位のサブ加算器からキャリーがあるとして加算演算を行なったLookC742～746のキャリーを、5段のサブ加算器のキャリーとして6段のサブ加算器に出力する。逆に、SubC回路730から「キャリーなし」の信号がMainC回路720に与えられると、MainC回路720は、SubC回路735のキャリーを出力する。

【0059】このようにサブ加算器として分割していない場合、14ビットの加算は0ビットからのキャリーを持って加算する必要があるため、14段の回路分の時間が必要であるが、複数に分割すると、0ビットからのキャリーは、SubC727～730、MainC回路720、SlaveC回路721～725の10段分の処理時間で済む。

【0060】たとえば「000101011001101」と「010000011110010」とを加算する場合について説明する。SubC回路727は、0ビット目同士の加算(1+0)を行なう。加算結果(1)はアダー回路701に供給される。また、キャリー(0)は、SubC回路728およびアダー回路702に供給される。アダー回路701は、SubC回路727から受取った値を0ビット目の加算結果として出力する。

【0061】SubC回路728は、1ビット目の値(0および1)と、0ビット目からのキャリー(0)との加算(0+1+0)を行なう。加算結果(1)はアダー回路702に供給される。また、キャリー(0)は、SubC回路729およびアダー回路703に供給される。アダー回路702は、0ビット目のキャリー(0)と2ビット目の加算結果(1)とを加算して、1ビット目の加算結果(1)を出力する。同様に、アダー回路703～715では、前のビットのキャリーと着目しているビットの加算結果との加算が行なわれる。

【0062】LookC回路742は、4段のサブ加算器でキャリーが発生したと仮定して4ビット目のキャリーの計算を行なう。すなわち、4ビット目の値同士の加算とキャリー(1)との加算(0+1+1)が実行される。演算の結果のキャリーはLookC回路743に与えられる。LookC回路747においても同様の処理が行なわれる。

【0063】LookC回路743は、5ビット目の値とLookC回路742からのキャリーとの加算演算(0+1+1)を実行し、演算の結果のキャリー(1)をLookC744に供給する。以下、LookC回路744～746および748～752においても同様の処理が行なわれる。LookC回路746の出力は、MainC回路720に供給される。LookC回路752の出力は、MainC回路726に供給される。

【0064】SubC回路731は、5ビット目同士の加

算(0+1)を行なう。加算結果(1)は、アダー回路705およびSlaveC回路716に供給される。キャリー(0)は、SubC回路732に供給される。SubC回路736でも同様の処理が行なわれる。

【0065】SubC回路732は、6ビット目同士の加算と5ビット目のキャリー(0)との加算(0+1+0)を行なう。加算結果(1)は、アダー回路706およびSlaveC717に供給される。キャリー(0)は、SubC回路733に供給される。同様に、SubC回路733～735において、前のビットのキャリーと着目しているビットの値との加算が行なわれる。なお、SubC回路735における加算結果はアダー回路709のみに供給され、SubC回路735におけるキャリーは、MainC回路720に供給される。SubC回路736～741においても同様の処理が行なわれる。

【0066】SlaveC回路716は、3ビット目のキャリー(0)と、4ビット目の加算結果(1)とを加算し、キャリーをアダー回路706およびSlaveC回路717に供給する。SlaveC回路717～719および721～725においても同様の処理が行なわれる。

【0067】MainC回路720は、SubC回路730が出力する前段のサブ加算器のキャリーが1の場合には、LookCの出力をSlaveC回路721に供給し、キャリーが0の場合には、SubC回路735の出力するキャリーをSlaveC回路721に供給する。MainC回路726も同様の処理を行なう。

【0068】以上説明したようにして、15ビットのデータ同士の加算が行なわれる。すなわち、SubC回路、LookC回路およびMainC回路で計算が行なわれた後、SlaveC回路およびアダー回路で計算が行われる。このように二段階にわけて加算処理が実行される。

【0069】例に挙げた15ビットではあまり大きな段数の差は発生していないが、もっと大きなビットの加算器たとえば130ビットでは、分割しない場合は129段分の処理時間がかかるのに対し、分割した場合は30段分の処理時間で済む。また、6段のサブ加算器のキャリーに由来する0ビットからのキャリーを求めるための回路の段数を5段(SubC回路727～730、MainC回路720の5段)とし、6段のサブ加算器内のキャリーを求めるための回路の段数を6段(SubC回路736～741とLookC回路747～752との2通りとも6段)としている。このような構成にすることにより、もっと大きなビットの加算器でも段数の違いがほとんどない構成にすることができる。これにより、タイミングなどに対する回路設計が容易になり、回路規模も小さくできるという効果がある。

【0070】なお、130ビットのパイプラインでの加算器では、4段、5段、6段、7段、8段、9段、10段、11段、12段、13段、14段、15段および16段のサブ加算器により加算器を構成するのが適当であ

る。

【0071】[べき乗剰余演算処理について] 図3は、べき乗剰余演算処理のフローチャートである。図4は、信号のタイミングを示すタイムチャートである。

【0072】従来は、モンゴメリ法による乗算剰余演算を使用してべき乗剰余演算を行なう場合、鍵eのビットを検索し $e_j = 1$ の場合は、式(12)の $[Y = MR(X \cdot Y)]$ を実行し、 $e_j = 0$ の場合は、式(12)の $[Y = MR(X \cdot Y)]$ をスキップし実行しなかった。本実施の形態では、 e_j の値にかかわらず常に式(12)の $[MR(X \cdot Y)]$ を実行する。式(12)の演算結果をYレジスタ103に記憶させるときに e_j の値301をチェックし、 $e_j = 1$ の場合はYレジスタ書込信号304を出してYレジスタ103に演算結果を書込み、 $e_j = 0$ の場合はYレジスタ書込信号304を出さないでYレジスタ103に演算結果を書込まないようする。このように、常に式(12)を実行することにより、べき乗剰余演算時間を一定にすることができる。MR(X・Y)演算信号302は、“H”のとき演算中であることを示す。Yレジスタ書込信号303は、“H”のときYレジスタ103にデータを書込むことを示す。

【0073】以上のように、常に式(12)の演算を実行するようにしたため演算時間が鍵の値に関係なく一定になる。このため、タイミング攻撃に対する耐性を確保できるという効果がある。なお、タイミング攻撃とは、暗号文、鍵の長さによって、処理時間が変化することに着目し、暗号の解読を行うことを意味する。

【0074】[べき乗剰余演算以外の演算について] べき乗剰余演算器には、べき乗剰余制御回路109、モンゴメリ乗算剰余・剰余制御回路110および加算・減算制御回路111が含まれる。このため、これらの制御回路を単独または組合わせて使用することにより、モンゴメリ乗算剰余演算、剰余演算、ストア演算、加算、減算、条件付加算、条件付減算など、各種演算を独立して実行することが可能である。

【0075】このようにべき乗剰余演算を構成する各種演算を独立して実行可能にしたことによりRSA暗号以外の各種暗号の演算を行なうことができるようになる。また、何らかの原因でべき乗剰余演算器が動作しない場合に、各種演算を独立して実行することにより、動作しない原因を究明することができる。

【0076】以上説明したように、本実施の形態では、頻繁に用いられる $[2B+N]$ の結果を保持するB2Nレジスタ105を設けることにより、モンゴメリ乗算剰余演算の高速化と演算回路の簡略化とを図ることができる。

【0077】[実施の形態2] 本発明の実施の形態2に係るべき乗剰余演算器は、実施の形態1に係るべき乗剰余演算器と同様の構成を有する。このため、その詳細な

説明は、ここでは繰返さない。

【0078】図5は、実施の形態2における信号のタイミングを示すタイムチャートである。本実施の形態では、モードレジスタ112に所定の値を設定することにより、べき乗剰余演算の実行方法を異ならせるものである。

【0079】スキップモード信号401が“H”の場合には、 $e_j = 0$ のときは式(12)を演算しないで式(13)の $[Y = MR(Y \cdot Y)]$ だけを演算する。スキップモード信号401が“L”の場合には、実施の形態1と同様の動作をする。

【0080】以上のように、式(12)の演算をスキップするモードを設定できるようにしたので、テスト時間の短縮を図ることができる。

【0081】[実施の形態3] 本発明の実施の形態3に係るべき乗剰余演算器は、実施の形態1に係るべき乗剰余演算器と同様の構成を有する。このため、その詳細な説明は、ここでは繰返さない。

【0082】図6は実施の形態3における信号のタイミングを示すタイムチャートである。通常の方法でべき乗剰余演算を行なっていくと、式(11)の演算結果は、

$$Y = MR(1 \cdot Y) = 1 \cdot R^2 \cdot R^{-1} \bmod N \\ = R \bmod N$$

となる。最初の $j = k$ から $e_j = 0$ の間、条件式のとおり式(12)ではYの値は変化しない。このため、式(13)の演算だけを考えると、

$$Y = MR(Y \cdot Y) = R \bmod N \cdot R \bmod N \cdot R^{-1} \bmod N \\ = R \cdot R \cdot R^{-1} \bmod N \\ = R \bmod N$$

となり、演算結果は同じである。以上のように $e_j = 1$ で式(12)の $[Y = MR(X \cdot Y)]$ を演算しYの値が変化するまでは式(13)の $[Y = MR(Y \cdot Y)]$ を演算してもYの値が変化しない。よって $e_j = 1$ になるまでは式(12)および式(13)の演算を実行しなくても構わないことがわかる。

【0083】通常は、演算時間を一定にするためにすべての演算を実行しているが、モードレジスタ112に値を設定することにより、サーチモード信号501を“H”にし、繰返し演算(式(12)および式(13))において、 $e_j = 1$ になるまでは式(12)および(13)の演算を実行しないで、 $[j-1]$ の演算のみ実行する。このように $e_j = 1$ か否かをチェックするモードを設けた。

【0084】以上のように、式(12)および(13)の演算をスキップするモードを設定できるようにしたので、鍵eの値によってはテスト時間の短縮が大幅に図れるという効果がある。

【0085】[実施の形態4] 本発明の実施の形態4に係るべき乗剰余演算器は、実施の形態1に係るべき乗剰余演算器と同様の構成を有する。このため、その詳細な

説明は、ここでは繰返さない。

【0086】図7は実施の形態4における信号のタイミングを示すタイムチャートである。本実施の形態では、実施の形態2で説明したスキップモードおよび実施の形態3で説明したサーチモードの両方のモードをモードレジスタ112に設定することができる。

【0087】たとえば、両方のモードを設定した場合には、 $e_j = 1$ になるまでは式(12)および式(13)を演算しない。一度 $e_j = 1$ になった後は式(12)および式(13)をともに演算する。その後は、 e_j の値によって、 $e_j = 0$ のときは、式(12)をスキップし式(13)のみ実行する。また、 $e_j = 1$ のときは式(12)および式(13)の両方の演算を実行する。

【0088】以上説明したように、スキップモードおよびサーチモードの両方のモードをモードレジスタ112に設定することにより、実施の形態2または実施の形態3よりもさらにテスト時間の短縮を図ることができる。

【0089】今回開示された実施の形態はすべての点で例示であって制限的なものではないと考えられるべきである。本発明の範囲は上記した説明ではなくて特許請求の範囲によって示され、特許請求の範囲と均等の意味および範囲内でのすべての変更が含まれることが意図される。

【0090】

【発明の効果】本発明によると、モンゴメリ乗算剰余演算に頻繁に用いられる値をレジスタに保持することにより、モンゴメリ乗算剰余演算を高速に実行することができる。

【図面の簡単な説明】

【図1】 べき乗剰余演算回路のハードウェア構成を示すブロック図である。

【図2】 演算回路内に設けられたパイプライン処理を実行する加算器のハードウェア構成を示すブロック図である。

【図3】 べき乗剰余演算処理のフローチャートである。

【図4】 信号のタイミングを示すタイムチャートである。

【図5】 信号のタイミングを示すタイムチャートである。

【図6】 信号のタイミングを示すタイムチャートである。

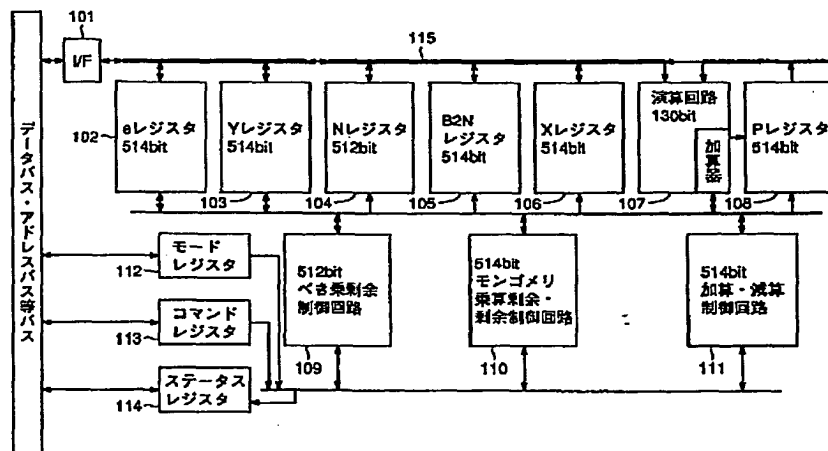
【図7】 信号のタイミングを示すタイムチャートである。

【図8】 従来のべき乗剰余演算処理のフローチャートである。

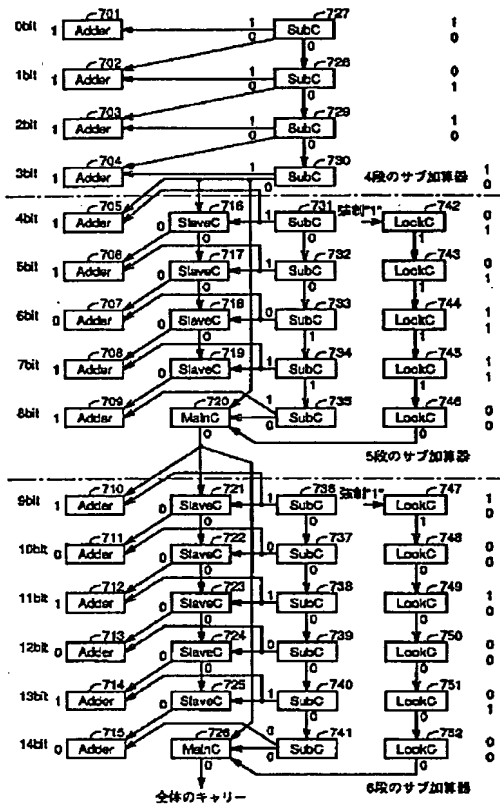
【符号の説明】

101 I/F回路、102 eレジスタ、103 Yレジスタ、104 Nレジスタ、105 B2Nレジスタ、106 Xレジスタ、107 演算回路、108 Pレジスタ、109 べき乗剰余制御回路、110 モンゴメリ乗算剰余・剰余制御回路、111 加算・減算制御回路、112 モードレジスタ、113 コマンドレジスタ、114 ステータスレジスタ、115 内部バス、701~715 アダー回路、716~719、721~725 SlaveC回路、720、726 MainC回路、727~741 SubC回路、742~752 LookC回路。

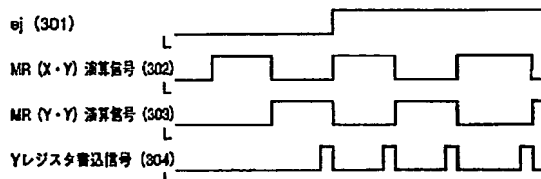
【図1】



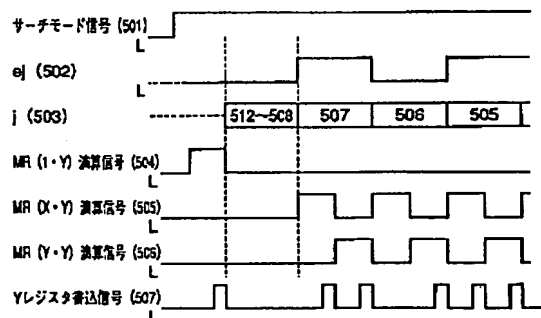
【図2】



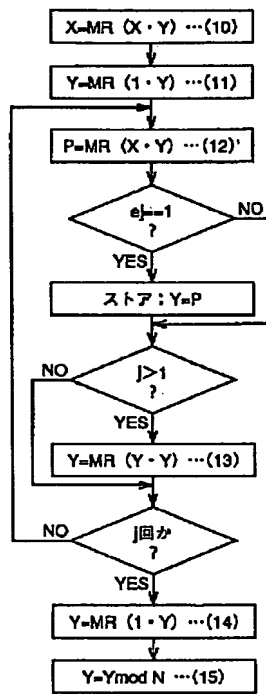
【図4】



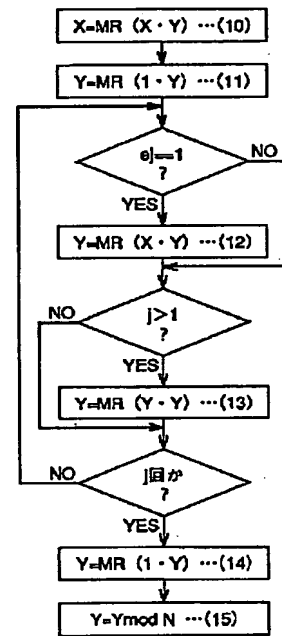
【図6】



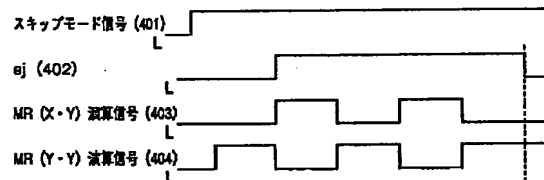
【図3】



【図8】



【図5】



【図7】

